iEFIS G4 Internals

General

This document describes the internals of a G4 iEFIS system. In particular it focuses on the development system that is used to create the root file system, operating system kernel and the EFIS application code itself (the exp4 file).

The parts to make a whole

The G4 system consists of:

1) A root file system. This is the typical Linux file system containing most of the usual folders and programs. The particular flavor of Linux is “Embedded Linux” which is similar to a desk top Linux without GUI. The many Linux commands are handled by a single executable called BusyBox. The root file system uses the EXT4 Linux journaling file system and resides on a SD Micro card on the G4 CPU board in partition 4.

2) The Linux kernel. This is a single file that contains the core of the operating system as well as many drivers. It is loaded on startup by the boot loader into memory and control is then handed to the kernel. The kernel requires two support files – a small environment file is located in /boot/extlinux which sets a few boot parameters. There are two copies – one is used if the kernel is running from the SD card on the G4 board, the other is used if the kernel has been loaded from the external SD card slot.
The second file is the device tree. This is loaded to memory by the bootloader as well.

The kernel file is named zImage and is located in /boot. The device tree “blob” is named stm32mp157c-dk2.dtb in the same folder. Note that the device tree file does not have the same contents as the original file which is destined for the dk2 development board.

3) The boot loader. The boot loader consists of two parts. The first part is a small program that is loaded directly by the fixed boot code contained in the CPU after reset. It’s main job is to initialize the DDR3 memory chip. It then loads the second part of the boot loader which is the well known u-boot. The first part of the boot loader is located in partition 1 of the SD card and a duplicate is placed in partition 2. Partition 3 contains the u-boot system.

U-boot has been extended to include a MGL system update and backup/restore facility. It is capable, among other things to build a working EFIS root file system onto a blank internal card from an external supplied disk image.
This u-boot extension is written in Embedded Pascal.

4) MGL iEFIS driver. Built into the kernel is a small driver that gives the EFIS code access to functions that can only be executed at kernel level. This driver is written using Embedded Pascal. The source is located under buildroot/output/build/linux-5.7.10/drivers/char/mgl with the Pascal source in the MGL driver project path (in Embedded Pascal).

5) M4 code. The CPU chip contains several processing cores. A cortex M4 core is used for almost all of the hardware interfacing by the EFIS firmware. The code for this is written in Embedded Pascal. It is loaded into memory reserved for the M4 on startup of the EFIS application program. The location of the source files is under the Embedded Pascal projects path “G4/iEFIS G4 M4”.

6) EFIS Application code. The EFIS application code is a single file named “exp4”. This file is located in the folder /efis. A script file located in /etc/init.d starts the exp4 application after the

kernel has completed its startup. The EFIS application can be halted after startup by sending an “X” via the Linux console – this enters a terminal command prompt.
The EFIS application code is written in Embedded Pascal but contains a few “C” source code files as well. Embedded Pascal is able to compile both Pascal and “C” code in a single application. The “C” code is used to provide the traditional “main” program entry to the Linux loader. It performs some GPU initialization and memory allocation before handing over to Pascal. The OpenGL GPU code has also been written in “C”.

The development system(s)

The Linux kernel, root file system and boot loader are built using the Linux build environment “Buildroot”. The Kernel version is 5.7.10. The buildroot for this version contains u-boot-2020.07. Packages as required are chosen from the defaults for this release with the exception of mesa3D the opensource version of OpenGL containing the Etnaviv driver for the Vivante GPU. The version used is mesa3d-23.1.4. This was done due to bugs in earlier versions.

The kernel is based on the STMicrosystems Linux platform for the STM32MP157 processor.

This buildroot release itself has several source code modifications, mostly related to drivers. Any development of EFIS code must be based on the modified code to work correctly.

The buildroot system runs under Linux. The iEFIS G4 development system is hosted on a Windows 11 PC running Linux under WSL2. Ubuntu 20.04 is used. Only the command line Linux is required. WSL2 can be used out of the box however as part of the WSL2 setup ensure that Windows Paths are NOT added to the Linux path variable. Windows Paths confuse the Buildroot system.

As part of the iEFIS development system release – buildroot is supplied in ready to use fashion in form of a “tar” archive. This needs to be installed under the users home folder resulting in the folder “buildroot”.

Compiling Linux and what goes with it

To work with buildroot is very simple: CD into the buildroot folder.

make menuconfig → brings up the overall Linux build configuration menu. You will likely not change anything here.

make linux-rebuild → this will compile the kernel and any driver changes

make uboot-rebuild → this will compile uboot (and also include any changes made by the Embedded Pascal u-boot extension code).

make → this creates the SDCard image you can use on the G4 CPU card.

Important buildroot folders

“build” → here you will find the sources of all selected packages and compiled files and object files. Each package gets its own folder suitably named. Note that Linux kernel and u-boot are just packages themselves.

“target” → this contains the root file system. Any file placed in here will find its way into the SD

card image.

Important notes

In the buildroot build system you have many options. Do a “#make help” to see what is available.

Do not execute any buildroot option that will reload source files as this will overwrite some of the sources that have been altered to work with the EFIS.

If you need to recompile a package and the buildroot system does not do it the best way is to go into the package folder in the build folder and delete all created object files yourself – this will force a recompile of that package only without buildroot extracting any source files from anywhere.

Embedded Pascal

The backbone of the EFIS application is the Embedded Pascal compiler.

The origins of Embedded Pascal date back to the 1990’s when the first compiler intended for the Z80 processor was released as shareware. It was subsequently released to support several different processors. Embedded Pascal was created by the same person that founded MGL Avionics. Naturally it was used to create the code for the first aviation products. Later the now well known ARM processors became popular and Embedded Pascal was adapted to support these. All of the more sophisticated EFIS systems from MGL use a flavor of ARM.

For the EFIS Embedded Pascal is used for the following code items:

1) The EFIS application itself – this is a very large application

2) The EFIS hardware interface code running on the M4 core

3) The u-boot extensions that allow easy system upgrades and backup/restore

4) A small Linux driver giving the EFIS access to functions normally reserved for the kernel.

5) A linux application emulating a LCD terminal (text output only). This is “g4te”. It is used during system startup to inform the user if the EFIS application program “exp4” cannot be found.

In most cases Embedded Pascal executes and compiles under Windows, inserting compiled object files straight into the relevant Linux folders as “shipped” object files. (Linux root system appears as a drive under Windows).

The actual EFIS code “exp4” is fully compiled under Windows into the “exp4” Linux executable.

It is possible to run Embedded Pascal under Linux using “Wine”. However this is not used here.

Obtaining the development system

At time of writing the development system is not yet available for general access. It is planned, should this be feasible and there is interest to release a “hello world” development system that can be used as a bases to develop applications from.

MGL may decide to release the EFIS source code at a later date excluding any parts that are subject to NDA.

Linux and the iEFIS

The Linux operating system as configured here is kept very basic. Nothing that is not required is included. In part this is done to speed up the boot process as much as is possible.

Linux is effectively only used to provide scheduling of tasks (a single system timer is used), provides file system support, USB support has been included as well as Network support. Support for LCD displays is included but the driver has been modified so it does not actually configure all hardware registers at every vertical blanking interval. Effectively this means we only are interested in the vertical blanking interrupt which is used by the GPU's own code. It is likely that in future updates this will be removed completely.

The Linux kernel here is designed to handle as little hardware as possible.

Fundamental to Linux is a console. This allows access to the command line. The console interface is a RS232 port on the EFIS. On classic EFIS systems the RS232 port 1 is used. On G4 native EFIS systems RS232 port 2 is used.

The port is always available as console port except if the EFIS has been configured to use it for something else (it can be dedicated to any of the supported functions). If this is the case control of the console port is taken away from Linux at start of the exp4 EFIS application.

If the console port is available and the EFIS code is running, sending an ASCII "X" to the console port causes the EFIS code to shut down and the Linux console will activate.

It is also possible to access an internal memory dump facility in the EFIS code via the console that remains active (provided it is not used for another purpose).

You can for example dump memory contents using "D 12345 100". This will dump 256 bytes in hexadecimal from address 0x12345. Note that this is a virtual address. The dump will also tell you the physical address of the location. This can be used to quickly check the contents of a variable etc. Embedded Pascal has a facility that shows the address of a variable you are interested in.

The console port operates at 115200Baud, 8 data bits, 1 start, 1 stop bit. A VT100 terminal is expected. A typical free terminal program that works in Windows is TeraTerm.

The EFIS application needs to interact with hardware. This is an area Linux is not very good at. However the CPU chip used has a additional cortex M4 core that is not subject to the Linux virtual memory model and thus operates in real address mode. It has access to the entire address space. It is possible to run code on this M4 completely in parallel to Linux and independent from it. This is an ideal way to handle the many hardware related tasks. The M4 handles all hardware interfacing from LCD display, Video decoding, serial ports, CAN bus and much more.

A 16Mbyte fixed, non-cached memory region is created on start up to act as data interface between the EFIS/Linux side and the M4. This is used mainly to hold communications buffers. In addition the CPU chip contains several regions of internal static R/W memory that is very fast – this is also used to share data and instructions between the two sides.

The code for the M4 is loaded at startup of the exp4 EFIS code. The code is included in the exp4 executable.

Editing text files

For simple viewing of text files in the Linux console use “cat”. For editing of text files in the Linux console use “nano”. Text files under Linux are usually used to create executable scripts.

Dealing with the real EFIS

Developing code is not complete without some means of executing it on the target platform in a convenient manner.

For the G4 we use a very nice way of creating a virtual disk on the PC and mounting this on the efis itself.

Embedded Pascal – with the EFIS loaded as project looks for a file called gcc.bat in the same folder as the project itself. This batch file is used to link the many object files into the final executable. At the end of this it copies the created “exp4” file to a folder called “export” under “C:”.

The export folder is shared as NFS folder on the network. There are several NFS servers that can be used under windows. We use Hanewin – although not free it works well. The free ones tend to have issues.

In addition to this you need to install an RNDIS USB driver under windows – Windows has such a driver but sometimes there are issues where it incorrectly identifies a connection as serial port. Google the issue and how to fix it if this affects you.

On the G4 EFIS you will find a USB micro connector inside the EFIS after removing the back panel. Connect a USB cable to a USB host port on your PC (at least USB 2.0).

On your EFIS configure your RNDIS network connection for a fixed IP address of 192.168.10.2. The EFIS will be 192.168.10.1

Connect a terminal to the EFIS console port, hit “X” to stop the EFIS application if it is running. You should see the “#” console prompt.

Type “ether”

This executes a script located in /bin. It will load and configure the RNDIS gadget driver on the EFIS.

If this succeeds you can test from PC DOS command prompt: ping 192.168.10.1 and from the Linux prompt: ping 192.168.10.2.

If this works your ethernet via USB connection between PC and EFIS is up and running.

Type “emnt”

This executes another script located in /bin. It will attempt to mount the export folder on the PC to /nfs on the EFIS.

CD into /nfs (on the EFIS)

type ls

You should see the files in your PC export folder. If you compiled the exp4 file it should be there.

Type ./exp4 → This will execute your exp4 file.

You can repeat “X” and “./exp4” as many times as you like during development. It is a quick and easy way to test your software. Note that the “debug()” procedure in the EFIS will print your desired text to the console which is very handy tool for debugging purposes.

Linux and kernel console messages

By default Linux startup and console messages on the console are suppressed by means of the environment file in /boot/extLinux. Loglevel is set to 1. To show messages set the Loglevel to 7. You can use nano as text editor – it is installed by default.

Note that there are three environment files:

extlinux.conf – this is the “normal” file name. It is not used.

extlinux.conf0 – this is used if the system is booting from the internal SD card

extlinux.conf1 – this is used if the system is booting from the external SD card

This is needed for u-boot so it hands over the correct location of the root file system.

CPU core usage

The STM32MP157 MPU contains effectively 4 CPUs

Two CPU are Cortex A7 processors. Linux is configured to run both of these cores. Linux will distribute tasks/threads between the two cores.

Currently the EFIS application only executes on one of the cores, performance being sufficient. The second core is largely dormant but may be utilized in future releases.

A single Cortex M4 is used to interface between Linux and the EFIS hardware. Linux itself is not involved with this core – it is entirely under control of the EFIS firmware.

A Vivante GC400 graphics processor is included (GPU). This is utilized via the open source OpenGL mesa3D implementation and a open source implementation of the Vivante driver – suitably called Etnaviv.

On the EFIS the GPU is used to draw the 3D terrain, runways and is also involved in the color decoding transformation for video streams.

Boot code source

The EFIS can boot either from the internal SD card using the first stage bootloader in the first card partition as well as u-boot contained in partition 3 or it can do so by reading the same code from a QUAD SPI NOR flash memory chip on the G4 CPU card.

This is determined by solder links on the CPU card. B0 is always closed, B1 is open. B2 closed will boot from SD card. B2 open will boot from the NOR flash memory (default).

Normally the system will boot from the flash memory. Due to the u-boot extension allowing restore of a disk image to an internal SD card it means it is possible to insert a new, blank SD card into the G4 card slot and then use the boot loader to copy an existing disk image into that card.

/bin/mkspinorboot is a script file that will copy partition 1 and 3 (first stage boot and u-boot) from the SD card to the SPI flash memory. This is normally only done once at the factory

when the flash memory chip initially is blank.

More details on the boot system in the document G4Boot.pdf from the MGL Avionics website.

SDCard.img

Buildroot during execution of “Make” creates the file SDCard.img in buildroot/output/images

This file is approximately 3.6GByte in size and includes the complete root file system as well as all the EFIS files required for a first installation. (Including the terrain data files which are large).

You can copy this file to a blank SD card using a disk imager such as WinImage32 under windows or dd in Linux. Balena-Etcher is another nice application that can do this.

The root file system here does not contain the efis folders in their final form but all the files are present.

On first run of this image a few things happen:

The image resizes partition 4 (which contains the root file system) to a size of 15GByte (even if the SD card is larger). At least a 16GByte card is needed.

This takes a few minutes – you should see the text “resizefs” on the LCD display during this time. Once done it will be replaced with “cannot find exp4…”.

All this is done using a script file which also builds the folder structure for the EFIS and moves the required files into their final locations. This cannot be done in the buildroot target image as these folders are created as case insensitive folders and this is only possible with an empty folder – also this requires changes to the Linux kernel which are not mainstream defaults so it cannot be done on the PC side in Linux..

The first stage boot loader and u-boot are also automatically copied into the flash memory from the SD card.

Once all these preparatory steps have completed the script file containing these instructions self-destructs.

A backup copy of the script file is located in /boot and is named S90resizedisk. It needs to be in /etc/init.d to execute on startup. If the u-boot extension code is tasked to copy an image file to internal disk it will attempt to copy this file from /boot to /etc/init.d.
If the SDCard.img from buildroot is installed the script file is already in the correct location.

The only task that remains is to install the exp4 file which can be performed using the boot loader (see document G4Boot.pdf).

The iEFIS simulator

The iEFIS simulator is complied using Delphi 5. It will also compile with minor changes in later versions. The Delphi project compiles the same sources as are used also in the Embedded Pascal compiler to create the actual exp4 executable. Additional files are compiled to create the GUI and related simulated devices etc. Where needed within the sources that are compiled by both compilers special comment structures are used to hide/expose certain code where there are differences.

This method ensures a very high degree of similarity between the simulated EFIS and the real

one. The iEFIS simulator is also the primary development environment – code is first developed on the simulator and once verified is then compiled for the real EFIS and tested.